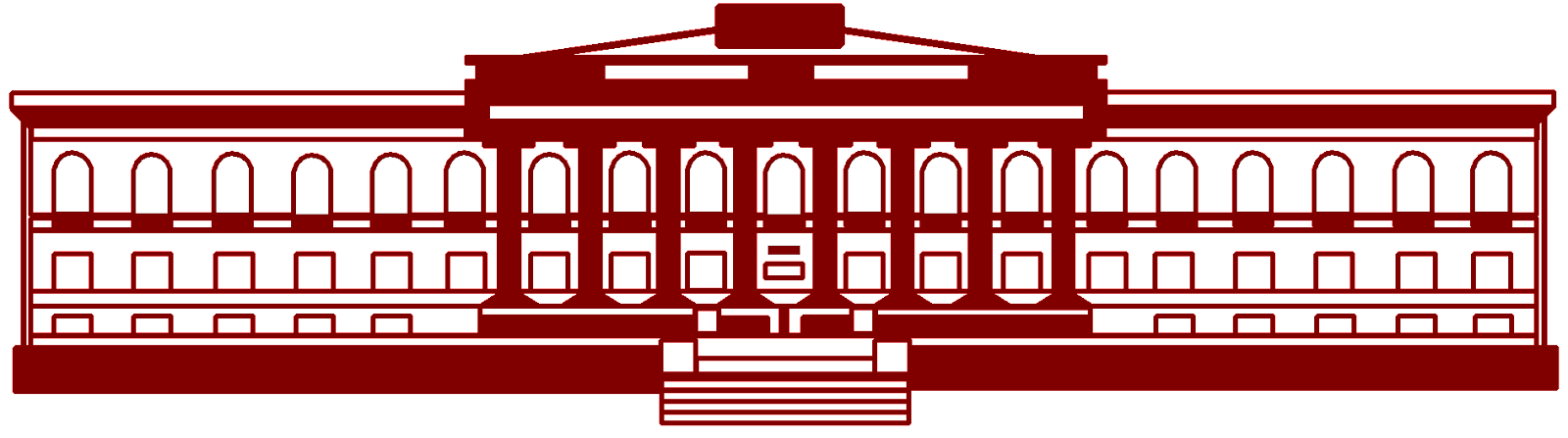
**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**імені ТАРАСА ШЕВЧЕНКА**

Факультет інформаційних технологій

**Кафедра прикладних інформаційних систем**



**Звіт**

до виконання лабораторної роботи №4

з дисципліни «**Веб-технології і програмування сайтів**»

на тему:

« Динамічне управління HTML-елементами засобами мови

JavaScript »

**Виконано:**

студ. групи ПП- 31, підгрупа 2

Шкандюк Анною Леонідівною

**Перевірено:**

Пирог М.В.

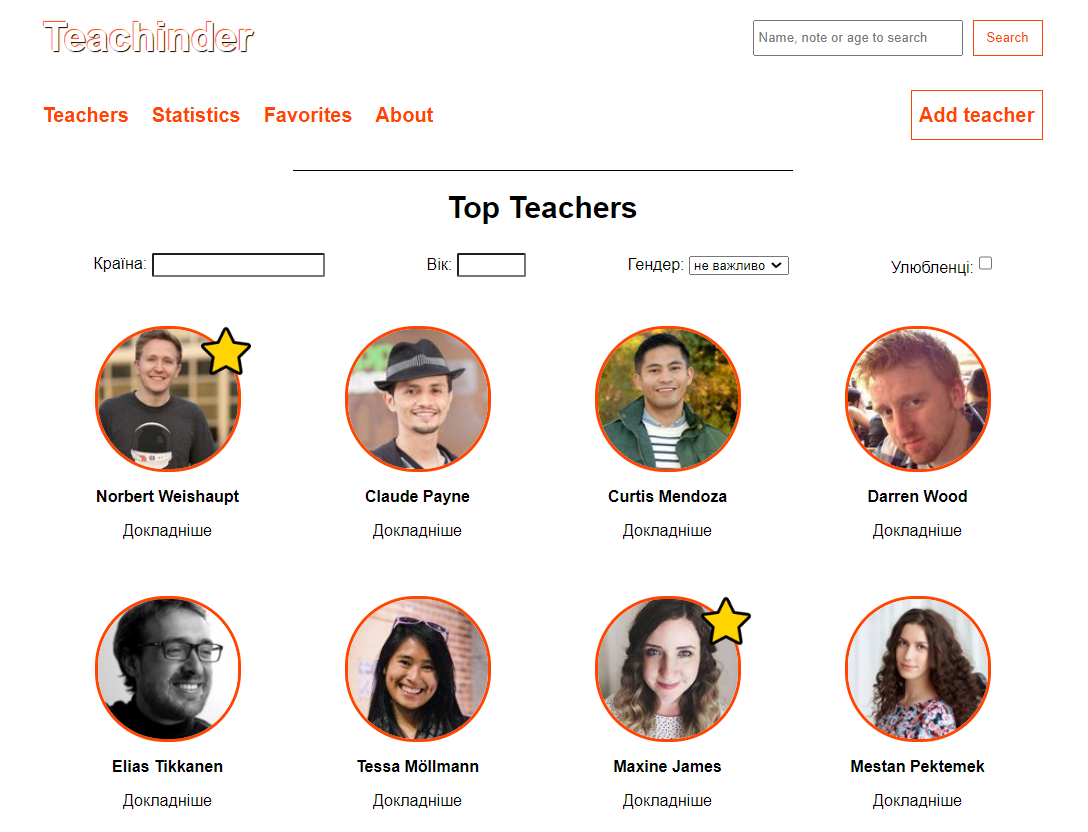
**Київ – 2023**

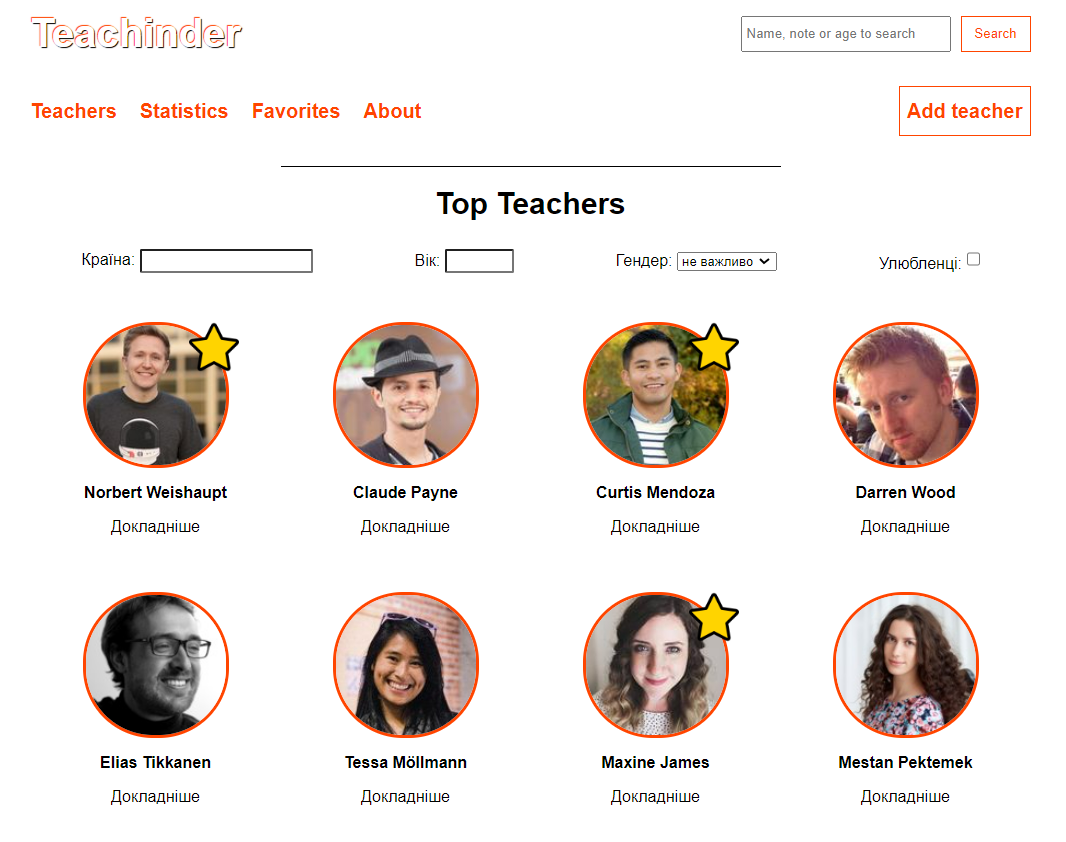
1. **Мета роботи:**

Навчитися застосовувати можливості мови JavaScript для динамічної зміни вигляду та поведінки HTML-елементів у відповідь на дії користувача.

1. **Завдання та їх виконання:**

**Завдання 1**. Відобразити масив об’єктів викладачів отриманий у лабораторній роботі №3 на html сторінці з лабораторної роботи №2 та реалізувати функціональність перегляду інформації про викладача та додавання у список вибраних (favorites).

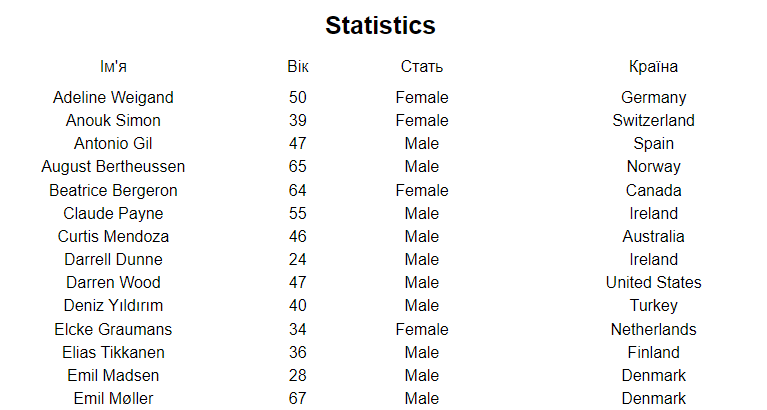


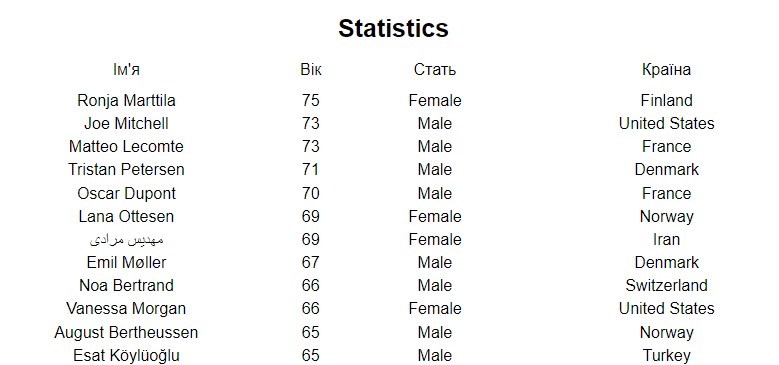


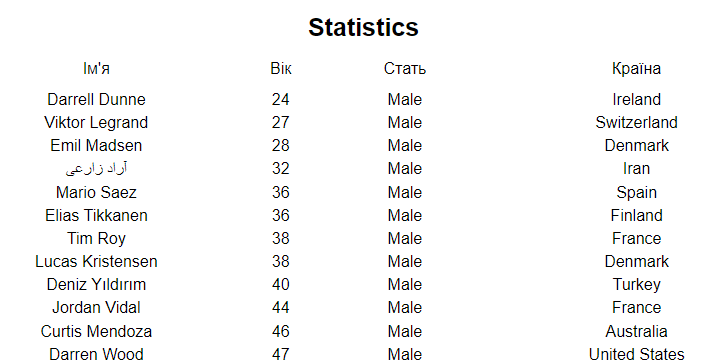
**Завдання 2.** Додати на html сторінку можливість фільтрації викладачів на сторінці по країні, віку, статі та тих, що є у списку вибраних (country, age, gender, favorite.

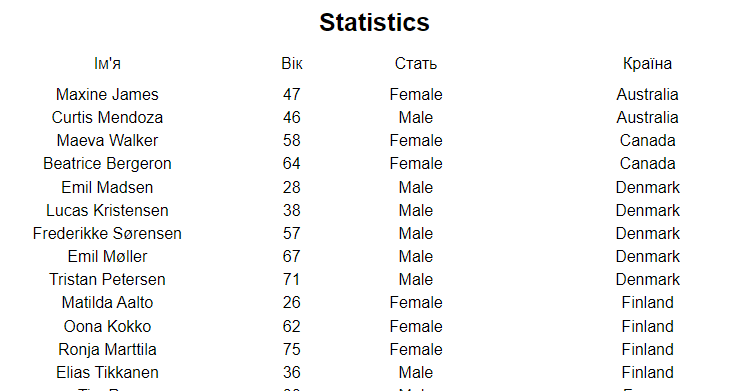


**Завдання 3.** Додати на html сторінку до блоку статистики можливість сортування за ім’ям, спеціальністю, країною, та віком (full\_name, course, age, b\_day, country). Змінювати сортування по кліку на заголовок таблиці.

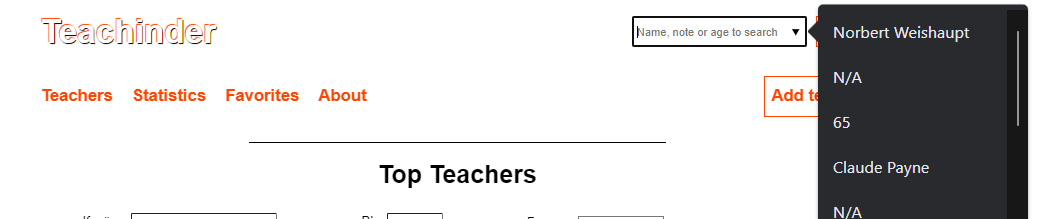




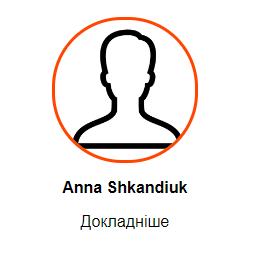
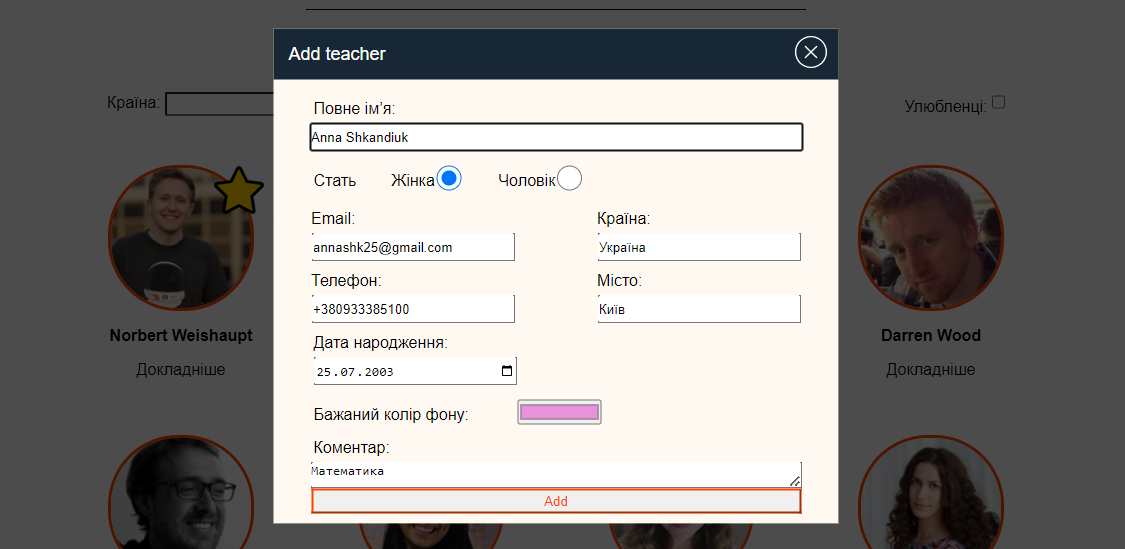


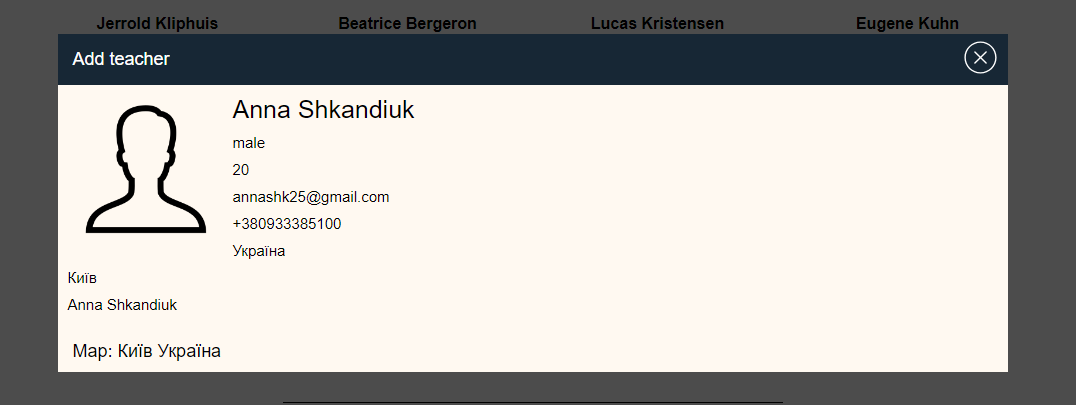


**Завдання 4** Додати на html сторінку функціональність пошуку по викладачах за параметрами: ім’я, коментар та вік (name, note, age)



**Завдання 5.** Реалізувати функціонал форми додавання викладача (teach\_add\_popup)





1. **Висновки до роботи**

За результатами проробленої роботи можемо сказати, що навчилася застосовувати можливості мови JavaScript для динамічної зміни вигляду та поведінки HTML-елементів у відповідь на дії користувача.

1. **Контрольні запитання**
2. Що таке DOM?

DOM (Document Object Model) - це представлення структури документа (зазвичай HTML або XML) як дерева об'єктів, доступне для маніпулювання програмами або скриптами. DOM дозволяє звертатися до кожного елемента документа і змінювати його властивості, структуру та вміст.

Основні характеристики DOM:

1. Структура дерева: DOM представляє документ як дерево, де кожен вузол (наприклад, елемент HTML, текстовий вміст, атрибут) є об'єктом. Кореневий вузол зазвичай є `<html>`, а з нього виходять гілки до всіх інших вузлів.

2. Зручний доступ: DOM надає можливість звертатися до елементів документа за допомогою об'єктів та методів. Наприклад, ви можете змінювати текстовий вміст, атрибути, стилі, додавати та видаляти елементи.

3. Інтерактивність: Ви можете використовувати JavaScript для динамічної зміни вмісту та поведінки сторінки на основі подій (натискання кнопок, наведення мишки тощо).

4. Кросбраузерність: DOM є стандартом, який підтримується більшістю сучасних браузерів. Це означає, що ви можете використовувати однаковий код для маніпулювання документами в різних браузерах.

5. Розділення структури та вигляду: DOM відділяє структуру документа (HTML-код) від його візуального відображення (CSS). Це дозволяє змінювати вигляд документа без зміни його структури.

6. Можливості обробки даних: DOM також дозволяє використовувати форми та елементи для збору даних від користувача та взаємодії з серверами.

Загалом, DOM є потужним інструментом для маніпулювання веб-сторінками та забезпечує динамічність та інтерактивність веб-додатків.

1. Які елементи html розмітки відображені в DOM, а які ні?

Більшість стандартних HTML-елементів відображаються в DOM (Document Object Model) як об'єкти, що дозволяє вам маніпулювати ними за допомогою JavaScript. Однак не всі елементи мають однозначне представлення в DOM, і деякі з них мають специфічні характеристики.

Елементи, які відображаються в DOM:

1. Більшість стандартних HTML-елементів, таких як `<div>`, `<p>`, `<span>`, `<h1>`, `<ul>`, `<li>`, тощо.

2. Форми та їхні елементи: `<form>`, `<input>`, `<select>`, `<textarea>`, тощо.

3. Таблиці та їхні елементи: `<table>`, `<tr>`, `<td>`, `<th>`, тощо.

4. Зображення: `<img>`.

5. Посилання: `<a>`.

6. Відео та аудіо: `<video>`, `<audio>`.

7. Скрипти та стилі: `<script>`, `<style>`.

Елементи, які можуть не мати однозначного відображення в DOM:

1. `<head>`: Це частина документа, яка містить метадані, стилі та посилання на зовнішні ресурси. Вона не відображається як окремий вузол в DOM, але її вміст доступний через `document.head`.

2. Текстовий вміст: Текст, який не обгорнутий в жоден HTML-елемент, називається текстовим вмістом. Він міститься в DOM, але як дочірній вузол іншого елемента (наприклад, `<body>` або блочних елементів).

3. Пробіли та переноси рядка: Пробіли та переноси рядка також можуть не мати однозначного відображення в DOM. Вони можуть бути враховані як текстовий вміст, але браузери можуть розглядати їх по-різному в різних ситуаціях.

Загалом, більшість стандартних HTML-елементів мають однозначне відображення в DOM, але існують деякі виключення та особливості, які варто враховувати при роботі з DOM.

1. Які елементи html розмітки відображені на сторінці, а які ні?

На сторінці HTML відображаються більшість стандартних HTML-елементів, а також є деякі елементи, які використовуються для забезпечення певного функціоналу, але не мають візуального представлення на сторінці. Ось кілька прикладів:

Елементи, які відображаються на сторінці:

1. Блокові елементи: `<div>`, `<p>`, `<h1>`, `<ul>`, `<li>`, `<table>`, тощо. Вони використовуються для організації структури та вмісту сторінки.

2. Інлайнові елементи: `<span>`, `<a>`, `<strong>`, `<em>`, `<img>`, тощо. Вони використовуються для надання спеціального стилю або функціоналу певним частинам тексту або контенту.

3. Форми та їхні елементи: `<form>`, `<input>`, `<select>`, `<textarea>`, тощо. Вони дозволяють користувачам взаємодіяти зі сторінкою, вводити дані та відправляти їх на сервер.

4. Зображення: `<img>`. Вони використовуються для вставки зображень на сторінку.

5. Посилання: `<a>`. Вони дозволяють створювати гіперпосилання на інші сторінки або ресурси.

6. Мультимедійні елементи: `<video>`, `<audio>`. Вони використовуються для вставки відео та аудіо на сторінку.

Елементи, які не мають візуального відображення:

1. `<meta>`: Метатеги використовуються для надання метаінформації про документ, такої як кодування, опис, ключові слова тощо. Вони не мають візуального відображення на сторінці, але важливі для пошукових систем та інших процесів.

2. `<link>`: Елемент `<link>` використовується для підключення зовнішніх файлів стилів, фавіконок та інших ресурсів.

3. `<script>`: Елемент `<script>` використовується для підключення зовнішніх скриптів або вставки JavaScript-коду на сторінку.

4. `<style>`: Елемент `<style>` використовується для вкладеного опису стилів.

5. `<noscript>`: Елемент `<noscript>` надає альтернативний контент, який відображається, якщо веб-переглядач не підтримує виконання JavaScript.

6. `<template>`: Елемент `<template>` використовується для зберігання клонованого контенту, який не відображається на сторінці, але може бути використаний для подальшого вставлення.

1. Який глобальний об’єкт використовується в JS для взаємодії з DOM.

У JavaScript глобальний об'єкт, який використовується для взаємодії з DOM (Document Object Model), називається `document`. Цей об'єкт представляє поточний HTML-документ та надає доступ до всіх елементів, властивостей та методів, які дозволяють маніпулювати вмістом та структурою документа.

Деякі з основних властивостей та методів глобального об'єкта `document` включають:

1. `document.getElementById(id)`: Повертає елемент за його ідентифікатором.

2. `document.querySelector(selector)`: Повертає перший елемент, який відповідає CSS-селектору.

3. `document.querySelectorAll(selector)`: Повертає список всіх елементів, які відповідають CSS-селектору.

4. `document.createElement(tagName)`: Створює новий HTML-елемент за назвою тегу.

5. `document.createTextNode(text)`: Створює новий текстовий вузол.

6. `element.appendChild(child)`: Додає дочірній вузол до елемента.

7. `element.innerHTML`: Отримує або встановлює HTML-вміст елемента.

8. `element.style`: Отримує або встановлює стилі елемента.

9. `element.addEventListener(event, callback)`: Додає обробник події до елемента.

10. `element.remove()`: Видаляє елемент з DOM.

1. Які Ви знаєте функції для пошуку елементів в DOM?

У JavaScript існують різні методи для пошуку елементів в DOM за допомогою селекторів, ідентифікаторів, класів тощо. Ось декілька з них:

1. `getElementById(id)`: Пошук елемента за ідентифікатором (id).

const element = document.getElementById("myElementId");

2. `querySelector(selector)`: Пошук першого елемента, який відповідає CSS-селектору.

const element = document.querySelector(".myClass");

3. `querySelectorAll(selector)`: Пошук всіх елементів, які відповідають CSS-селектору.

const elements = document.querySelectorAll("div");

4. `getElementsByClassName(className)`: Пошук елементів за класом.

const elements = document.getElementsByClassName("myClass");

5. `getElementsByTagName(tagName)`: Пошук елементів за тегом (назвою тегу).

const elements = document.getElementsByTagName("p");

6. `getElementsByName(name)`: Пошук елементів за атрибутом `name`.

const elements = document.getElementsByName("myName");

Ці методи дозволяють знаходити елементи в DOM на основі різних критеріїв. Коли елементи знаходяться, ви можете маніпулювати їх властивостями, структурою та вмістом за допомогою JavaScript.

1. Що повертають функції: querySelector, querySelectorAll?

Функції `querySelector` та `querySelectorAll` є методами глобального об'єкта `document` у JavaScript, які використовуються для пошуку елементів в DOM за допомогою CSS-селекторів. Ось що ці функції повертають:

1. `querySelector(selector)`:

Ця функція приймає CSS-селектор як аргумент і повертає перший елемент, який відповідає цьому селектору. Якщо відповідних елементів не знайдено, функція поверне `null`.

Приклад використання:

const element = document.querySelector(".myClass");

2. `querySelectorAll(selector)`:

Ця функція також приймає CSS-селектор як аргумент і повертає колекцію (NodeList) всіх елементів, які відповідають цьому селектору. Якщо відповідних елементів не знайдено, функція поверне пусту колекцію (NodeList).

Приклад використання:

const elements = document.querySelectorAll("div");

Якщо ви використовуєте `querySelector`, буде повернуто лише перший відповідний елемент. У випадку `querySelectorAll`, буде повернуто всі відповідні елементи у вигляді колекції (NodeList), яку можна перебирати за допомогою циклу або методів масивів (якщо потрібно).

1. Як створити html елемент за допомогою JS?

У JavaScript ви можете створити нові HTML-елементи та додати їх до DOM за допомогою методу `createElement()` глобального об'єкта `document`. Ось як це зробити:

1. Спершу створіть новий елемент за допомогою `createElement(tagName)`, де `tagName` - це назва тегу нового елемента.

2. Встановіть необхідні властивості та атрибути для цього нового елемента.

3. Додайте створений елемент до DOM за допомогою методів, таких як `appendChild()` для додавання в дочірні елементи і `insertBefore()` для вставки перед іншим елементом.

Ось приклад створення нового `<div>` елемента та його додавання до DOM:

// Створення нового <div> елемента

const newDiv = document.createElement("div");

// Встановлення текстового вмісту та класу для нового елемента

newDiv.textContent = "Це новий елемент";

newDiv.className = "my-class";

// Додавання нового елемента до DOM (до кореневого елемента <body>)

document.body.appendChild(newDiv);

У цьому прикладі ми створили новий `<div>` елемент, встановили йому текстовий вміст та клас, а потім додали його до кінця кореневого елемента `<body>`.

1. Що таке властивості innerHTML, outerHTMLта innerText? Яка між ними різниця?

`innerHTML`, `outerHTML` та `innerText` - це властивості елементів DOM у JavaScript, які використовуються для отримання або встановлення вмісту та структури елементів. Однак між ними є певні різниці у їхньому призначенні та поведінці:

1. `innerHTML`:

- Властивість `innerHTML` дозволяє отримувати або встановлювати HTML-вміст елемента.

- При отриманні властивості `innerHTML` ви отримуєте рядок, який містить HTML-вміст всередині елемента.

- При встановленні значення властивості `innerHTML` ви можете задати новий HTML-вміст для елемента, і старий вміст буде замінений.

- Наприклад:

const divElement = document.querySelector("div");

const htmlContent = divElement.innerHTML; // Отримання HTML-вмісту

divElement.innerHTML = "<p>Новий вміст</p>"; // Встановлення нового HTML-вмісту

2. `outerHTML`:

- Властивість `outerHTML` дозволяє отримувати або встановлювати HTML-представлення самого елемента, включно з ним самим.

- При отриманні властивості `outerHTML` ви отримуєте рядок, який містить повний HTML-код елемента, включаючи його власний тег.

- При встановленні значення властивості `outerHTML` елемент буде повністю замінений на новий HTML-код, який ви вказали.

- Наприклад:

const divElement = document.querySelector("div");

const outerHtmlContent = divElement.outerHTML; // Отримання HTML-представлення елемента

divElement.outerHTML = "<p>Новий елемент</p>"; // Заміна елемента новим HTML-кодом

3. `innerText`:

- Властивість `innerText` дозволяє отримувати або встановлювати текстовий вміст елемента, враховуючи тільки видимий текст, а не HTML-структуру.

- При отриманні властивості `innerText` ви отримуєте звичайний текст, який відображається всередині елемента.

- При встановленні значення властивості `innerText` ви можете задати новий текст для елемента, і старий текст буде замінений.

- Наприклад:

const divElement = document.querySelector("div");

const textContent = divElement.innerText; // Отримання текстового вмісту

divElement.innerText = "Новий текст"; // Встановлення нового текстового вмісту

Загалом, різниця між цими властивостями полягає в тому, який тип вмісту вони отримують або встановлюють (HTML або текст) та який обсяг елементів DOM вони включають.

1. Як видалити html елемент з розмітки?

Можна видалити HTML-елемент з розмітки DOM за допомогою методу `remove()` або `parentNode.removeChild(childNode)`. Ось як це зробити:

1. Метод `remove()`:

- Цей метод викликається на елементі, який ви хочете видалити, і видаляє цей елемент з DOM.

- Наприклад:

const elementToRemove = document.querySelector(".my-class");

elementToRemove.remove(); // Видалення елемента з DOM

2. Метод `parentNode.removeChild(childNode)`:

- Цей метод викликається на батьківському елементі та приймає в якості аргумента дочірній елемент, який ви хочете видалити.

- Наприклад:

const parentElement = document.querySelector(".parent");

const childElement = document.querySelector(".child");

parentElement.removeChild(childElement); // Видалення дочірнього елемента з батьківського елемента

1. Як скопіювати існуючий html елемент?

Для копіювання існуючого HTML-елемента в DOM ви можете використовувати метод `cloneNode()` разом з параметром `true`. Ось як це зробити:

const existingElement = document.querySelector(".existing-element");

const clonedElement = existingElement.cloneNode(true);

// Тепер clonedElement є копією existingElement

У методі `cloneNode(true)`, параметр `true` вказує, що ви хочете включити всі дочірні елементи, атрибути та вміст копійованого елемента. Якщо б ви передали `false`, то створилася би пуста копія без вмісту та дочірніх елементів.

Тепер `clonedElement` буде копією `existingElement`, і ви можете вставити цю копію до DOM за допомогою методу `appendChild()` або інших методів вставки.

1. Як додати новий елемент до існуючого?

Щоб додати новий елемент до існуючого елемента в DOM, ви можете використовувати методи, такі як `appendChild()`, `insertBefore()`, `innerHTML` та інші. Ось декілька способів додавання нового елемента до існуючого:

1. Метод `appendChild()`:

- Цей метод додає переданий елемент як дочірній елемент в кінець батьківського елемента.

- Наприклад:

const parentElement = document.querySelector(".parent");

const newElement = document.createElement("div");

parentElement.appendChild(newElement); // Додавання нового елемента до кінця батьківського елемента

2. Метод `insertBefore()`:

- Цей метод додає переданий елемент перед вказаним елементом у списку дочірніх елементів батьківського елемента.

- Наприклад:

const parentElement = document.querySelector(".parent");

const newElement = document.createElement("div");

const existingElement = document.querySelector(".existing");

parentElement.insertBefore(newElement, existingElement); // Додавання нового елемента перед існуючим елементом

3. Властивість `innerHTML`:

- Ви можете використовувати властивість `innerHTML` батьківського елемента для встановлення нового HTML-вмісту, включаючи і новий елемент.

- Наприклад:

const parentElement = document.querySelector(".parent");

parentElement.innerHTML += "<div>Новий елемент</div>"; // Додавання нового елемента за допомогою innerHTML

4. Властивість `insertAdjacentHTML()`:

- Ця властивість дозволяє вставити HTML-рядок перед, після, перед внутрішнім вмістом або внутрішнім вмістом вказаного елемента.

- Наприклад:

const existingElement = document.querySelector(".existing");

existingElement.insertAdjacentHTML("beforebegin", "<div>Новий елемент</div>"); // Додавання нового елемента перед існуючим елементом

Ці способи дозволяють додати новий елемент до існуючого елемента або до його вмісту в DOM. Вибір методу залежить від вашого конкретного випадку та потреб.

1. Які DOM події (events) існують для роботи з полями вводу (input, textarea, contenteditable тег)?

Для роботи з полями вводу, такими як `<input>`, `<textarea>` та елементи з атрибутом `contenteditable`, існує багато подій (events), які дозволяють взаємодіяти з вмістом та станом цих полів. Ось декілька з них:

1. `input`: Ця подія спрацьовує при введенні або зміні значення в поле вводу.

const inputElement = document.querySelector("input");

inputElement.addEventListener("input", (event) => {

console.log("Змінено значення:", event.target.value);

});

2. `change`: Ця подія спрацьовує після виходу з фокусу поля вводу після зміни значення.

const inputElement = document.querySelector("input");

inputElement.addEventListener("change", (event) => {

console.log("Змінено значення:", event.target.value);

});

3. `focus`: Ця подія спрацьовує при отриманні фокусу на полі вводу.

const inputElement = document.querySelector("input");

inputElement.addEventListener("focus", () => {

console.log("Отримано фокус");

});

4. `blur`: Ця подія спрацьовує при втраті фокусу полем вводу.

const inputElement = document.querySelector("input");

inputElement.addEventListener("blur", () => {

console.log("Втрачено фокус");

});

5. `keydown`, `keyup`: Ці події спрацьовують при натисканні та відпусканні клавіші на клавіатурі.

const inputElement = document.querySelector("input");

inputElement.addEventListener("keydown", (event) => {

console.log("Натиснута клавіша:", event.key);

});

inputElement.addEventListener("keyup", (event) => {

console.log("Відпущена клавіша:", event.key);

});

1. Які DOM події (events) існують для роботи з мишкою?

Для роботи з мишкою існують багато подій (events), які дозволяють взаємодіяти з різними аспектами мишкового взаємодії. Ось декілька з них:

1. `click`: Ця подія спрацьовує при кліку (одиночному натисканні та відпусканні) лівої кнопки миші.

const element = document.querySelector(".element");

element.addEventListener("click", () => {

console.log("Клікнуто");

});

2. `dblclick`: Ця подія спрацьовує при подвійному кліку лівої кнопки миші.

const element = document.querySelector(".element");

element.addEventListener("dblclick", () => {

console.log("Подвійний клік");

});

3. `mousedown`, `mouseup`: Ці події спрацьовують при натисканні та відпусканні будь-якої кнопки миші над елементом.

const element = document.querySelector(".element");

element.addEventListener("mousedown", () => {

console.log("Натиснуто кнопку миші");

});

element.addEventListener("mouseup", () => {

console.log("Відпущено кнопку миші");

});

4. `mousemove`: Ця подія спрацьовує при русі миші над елементом.

const element = document.querySelector(".element");

element.addEventListener("mousemove", (event) => {

console.log("Рух миші на позиції X:", event.clientX, "Y:", event.clientY);

});

5. `mouseenter`, `mouseleave`: Ці події спрацьовують, коли мишка потрапляє в межі елемента (mouseenter) або покидає межі елемента (mouseleave).

const element = document.querySelector(".element");

element.addEventListener("mouseenter", () => {

console.log("Мишка в елементі");

});

element.addEventListener("mouseleave", () => {

console.log("Мишка поза елементом");

});

1. Які DOM події (events) існують для взаємодії з html елементами?

Для взаємодії з HTML-елементами існує багато подій (events), які дозволяють відстежувати різні дії користувача та інші зміни в елементах. Ось декілька з них:

1. `click`: Подія спрацьовує при кліку на елементі.

2. `dblclick`: Подія спрацьовує при подвійному кліку на елементі.

3. `mouseenter`, `mouseleave`: Події спрацьовують, коли мишка входить в межі елемента (mouseenter) або покидає їх (mouseleave).

4. `mouseover`, `mouseout`: Події спрацьовують, коли мишка наведена на елемент (mouseover) або покидає його (mouseout), включаючи ситуацію, коли мишка наводиться на дочірні елементи.

5. `mousedown`, `mouseup`: Події спрацьовують при натисканні (mousedown) та відпусканні (mouseup) кнопки миші над елементом.

6. `focus`, `blur`: Подія focus спрацьовує при отриманні фокусу на елементі (наприклад, при кліку на поле вводу). Подія blur спрацьовує, коли елемент втрачає фокус.

7. `keydown`, `keyup`: Події спрацьовують при натисканні (keydown) та відпусканні (keyup) клавіші на клавіатурі.

8. `input`, `change`: Подія input спрацьовує при зміні значення поля вводу. Подія change спрацьовує після виходу з фокусу поля вводу після зміни значення.

9. `submit`: Подія спрацьовує при поданні форми (наприклад, при натисканні кнопки "Submit").

10. `scroll`: Подія спрацьовує під час прокручування вмісту елемента.

11. `contextmenu`: Подія спрацьовує при відкритті контекстного меню (правий клік) на елементі.

12. `load`: Подія спрацьовує, коли ресурс (наприклад, зображення) повністю завантажений.

13. `resize`: Подія спрацьовує при зміні розмірів вікна браузера або іншого елемента.

14. `drag`, `drop`: Події спрацьовують при перетягуванні (drag) та відпусканні (drop) елемента.

1. Якими методами можна додати подію (event) до html елемента?

Існує декілька способів додавання подій (events) до HTML-елементів в JavaScript. Ось декілька з них:

1. Властивість `on[event]`:

Ви можете встановити обробник події безпосередньо через властивість `onevent` елемента. Наприклад, для кліку:

const buttonElement = document.querySelector("button");

buttonElement.onclick = function() {

console.log("Клікнуто на кнопці");

};

Однак цей підхід має свої обмеження, особливо в тому, що ви можете встановити лише один обробник для кожної події, і якщо ви спробуєте встановити ще один, він замінить попередній.

2. Метод `addEventListener()`:

Цей метод дозволяє додати більше одного обробника події до елемента, а також надає більше гнучкості. Приклад додавання обробника для кліку:

const buttonElement = document.querySelector("button");

buttonElement.addEventListener("click", function() {

console.log("Клікнуто на кнопці");

});

Ви також можете використовувати `addEventListener()` для віддалення обробника події, якщо це необхідно:

buttonElement.removeEventListener("click", обробник);

3. Inline HTML атрибути:

Ви також можете використовувати inline атрибути HTML-елемента, такі як `onclick`, напряму в розмітці:

<button onclick="console.log('Клікнуто на кнопці')">Кнопка</button>

Однак використання inline атрибутів може змішувати HTML та JavaScript-логіку, що може бути складніше для обслуговування.

1. Як можна взаємодіяти з дочірніми елементами (child nodes)?

Для взаємодії з дочірніми елементами (child nodes) ви можете використовувати різні методи та властивості DOM. Ось декілька способів, які допоможуть вам працювати з дочірніми елементами:

1. Властивість `childNodes`: Ця властивість містить колекцію всіх дочірніх елементів елемента. Вона включає всі типи вузлів, такі як текстові, коментарі тощо.

const parentElement = document.querySelector(".parent");

const childNodes = parentElement.childNodes;

for (const node of childNodes) {

if (node.nodeType === Node.ELEMENT\_NODE) {

console.log("Дочірній елемент:", node);

}

}

2. Метод `children`: Цей метод повертає колекцію тільки дочірніх елементів, ігноруючи інші типи вузлів.

const parentElement = document.querySelector(".parent");

const children = parentElement.children;

for (const child of children) {

console.log("Дочірній елемент:", child);

}

3. Властивість `firstChild` та `lastChild`: Ці властивості дозволяють отримати перший та останній дочірні елементи вузла.

const parentElement = document.querySelector(".parent");

const firstChild = parentElement.firstChild;

const lastChild = parentElement.lastChild;

console.log("Перший дочірній елемент:", firstChild);

console.log("Останній дочірній елемент:", lastChild);

4. Властивість `firstElementChild` та `lastElementChild`: Ці властивості дозволяють отримати перший та останній дочірні елементи, ігноруючи інші типи вузлів.

const parentElement = document.querySelector(".parent");

const firstElementChild = parentElement.firstElementChild;

const lastElementChild = parentElement.lastElementChild;

console.log("Перший дочірній елемент:", firstElementChild);

console.log("Останній дочірній елемент:", lastElementChild);

5. Методи `appendChild()`, `insertBefore()`, `removeChild()`: Ці методи дозволяють додавати, вставляти та видаляти дочірні елементи.

const parentElement = document.querySelector(".parent");

const newElement = document.createElement("div");

// Додати новий елемент в кінець

parentElement.appendChild(newElement);

// Вставити новий елемент перед існуючим

const existingElement = document.querySelector(".existing");

parentElement.insertBefore(newElement, existingElement);

// Видалити дочірній елемент

parentElement.removeChild(existingElement);

1. Як можна взаємодіяти з батьківським елементом (parent node) та з сусідніми елементами (sibling nodes)?

Для взаємодії з батьківським елементом (parent node) та сусідніми елементами (sibling nodes) ви можете використовувати різні методи та властивості DOM. Ось декілька способів:

1. Властивість `parentNode`: Ця властивість дозволяє отримати батьківський елемент поточного вузла.

const childElement = document.querySelector(".child");

const parentElement = childElement.parentNode;

console.log("Батьківський елемент:", parentElement);

2. Властивість `parentElement`: Якщо ви працюєте з елементами, ви також можете використовувати властивість `parentElement`, яка також дозволяє отримати батьківський елемент.

const childElement = document.querySelector(".child");

const parentElement = childElement.parentElement;

console.log("Батьківський елемент:", parentElement);

3. Властивість `previousSibling` та `nextSibling`: Ці властивості дозволяють отримати попередній та наступний сусідні вузли.

const currentElement = document.querySelector(".current");

const previousSibling = currentElement.previousSibling;

const nextSibling = currentElement.nextSibling;

console.log("Попередній сусідній вузол:", previousSibling);

console.log("Наступний сусідній вузол:", nextSibling);

4. Властивість `previousElementSibling` та `nextElementSibling`: Ці властивості дозволяють отримати попередній та наступний сусідні елементи, ігноруючи інші типи вузлів.

const currentElement = document.querySelector(".current");

const previousElementSibling = currentElement.previousElementSibling;

const nextElementSibling = currentElement.nextElementSibling;

console.log("Попередній сусідній елемент:", previousElementSibling);

console.log("Наступний сусідній елемент:", nextElementSibling);

5. Властивість `parentNode.children`: Якщо вам потрібно отримати всіх дітей батьківського елемента (тільки елементи, без інших типів вузлів), ви можете використовувати властивість `children` батьківського елемента.

const parentElement = document.querySelector(".parent");

const children = parentElement.children;

for (const child of children) {

console.log("Дочірній елемент:", child);

}

1. Чим відрізняється HTML колекція від масиву і як колекцію можна перетворити в масив?

HTML колекція та масив - це два різні типи об'єктів у JavaScript, які використовуються для зберігання та управління наборами даних, але вони мають деякі відмінності.

1. HTML колекція:

- HTML колекція - це спеціальний тип об'єкта, який містить набір елементів DOM, які вибрані за певним критерієм (наприклад, за тегом, класом тощо).

- HTML колекція не є масивом, і, як правило, не має всіх методів та властивостей, які є у масиві (наприклад, методів `forEach`, `map`, `filter` тощо).

- HTML колекція оновлюється автоматично, якщо структура DOM змінюється.

2. Масив:

- Масив - це стандартний тип об'єкта у JavaScript, який має багато вбудованих методів та властивостей для маніпулювання та роботи з даними.

- Масиви зберігають дані у впорядкованій послідовності, і кожен елемент має індекс (числовий ключ).

- Масиви можуть містити будь-який тип даних, включаючи числа, рядки, об'єкти, функції тощо.

Якщо ви маєте HTML колекцію та хочете працювати з нею як з масивом, ви можете використовувати метод `Array.from()` для перетворення колекції у звичайний масив. Ось приклад:

const htmlCollection = document.getElementsByClassName("elements");

const arrayFromCollection = Array.from(htmlCollection);

// Тепер `arrayFromCollection` - це масив, і ви можете використовувати методи масиву

arrayFromCollection.forEach(element => {

console.log(element.textContent);

});

1. Як отримати інформацію, про застосовані стилі до елементу та як додати стиль до елементу?

Щоб отримати інформацію про застосовані стилі до елементу та додати стилі до елементу, ви можете використовувати методи та властивості, які надаються DOM та CSS в JavaScript.

1. Отримання інформації про стилі:

Ви можете використовувати властивість `window.getComputedStyle()` для отримання обчислених стилів для елементу. Цей метод повертає об'єкт `CSSStyleDeclaration`, який містить всі застосовані стилі.

const element = document.querySelector(".element");

const computedStyles = window.getComputedStyle(element);

// Отримання значення конкретної властивості

const backgroundColor = computedStyles.backgroundColor;

const fontSize = computedStyles.fontSize;

2. Додавання стилів до елементу:

Є декілька способів додавання стилів до елементу. Один із них - це використання властивості `style` елемента.

const element = document.querySelector(".element");

// Додавання стилю до елемента

element.style.backgroundColor = "blue";

element.style.fontSize = "18px";

Проте цей підхід прив'язує стилі безпосередньо до HTML коду, що може зробити код менш читабельним та важче для підтримки. Краще використовувати цей метод для динамічних змін стилів на сторінці.

Ще одним способом є додавання класів до елемента, а потім опис стилів для цих класів у CSS-файлі.

3. Застосування класів:

Ви можете додавати та видаляти класи до/з елемента за допомогою методів `classList.add()` та `classList.remove()`.

const element = document.querySelector(".element");

// Додавання класу

element.classList.add("blue-background");

// Видалення класу

element.classList.remove("blue-background");

У вас повинен бути стилізований клас у CSS для елементів з цими класами:

.blue-background {

background-color: blue;

}

1. Як взаємодіяти з атрибутами елементу? Як взаємодіяти з data атрибутами?

Взаємодія з атрибутами елементу та data-атрибутами здійснюється за допомогою методів та властивостей DOM. Ось як можна взаємодіяти з атрибутами та data-атрибутами:

1. Робота з атрибутами:

Для отримання та зміни значень атрибутів ви можете використовувати властивості елемента. Для отримання значення атрибуту використовуйте `.getAttribute()`, а для зміни - `.setAttribute()`.

const linkElement = document.querySelector("a");

// Отримання значення атрибуту

const hrefValue = linkElement.getAttribute("href");

// Зміна значення атрибуту

linkElement.setAttribute("href", "https://www.example.com");

2. Робота з data-атрибутами:

Data-атрибути - це спеціальні атрибути, які починаються з `data-` і використовуються для зберігання додаткової інформації про елемент, яку можна використовувати в JavaScript або CSS.

Для отримання та зміни значень data-атрибутів, ви також можете використовувати властивості елемента:

const element = document.querySelector(".element");

// Отримання значення data-атрибуту

const value = element.getAttribute("data-value");

// Зміна значення data-атрибуту

element.setAttribute("data-value", "new-value");

Однак, для зручності, DOM API також надає доступ до data-атрибутів через властивості об'єкта елемента:

const element = document.querySelector(".element");

// Отримання значення data-атрибуту через властивість

const value = element.dataset.value;

// Зміна значення data-атрибуту через властивість

element.dataset.value = "new-value";

Важливо відзначити, що властивість `dataset` дозволяє звертатися до всіх data-атрибутів елемента, включаючи вказівку їхніх імен у camelCase стилі (наприклад, `data-value` стає `element.dataset.value`).

1. Що таке вспливання (bubbling) події? Як його відмінити?

Вспливання події (event bubbling) - це механізм в JavaScript, коли подія спочатку виникає на найглибшому дочірньому елементі та потім "вспливає" вгору по ієрархії батьківських елементів. Це означає, що коли ви спрацьовуєте подію на дочірньому елементі, подія також буде автоматично спрацьовувати на всіх його батьківських елементах відповідно до їхньої ієрархії.

Наприклад, якщо у вас є така ієрархія елементів:

<div class="parent">

<div class="child">

<button class="button">Натисни мене</button>

</div>

</div>

І ви додаєте обробник події на кнопку:

const button = document.querySelector(".button");

button.addEventListener("click", function() {

console.log("Кнопка була натиснута");

});

Тоді, якщо ви клікнете на кнопку, обробник події буде викликаний не тільки для кнопки, але і для обох контейнерних елементів `.child` та `.parent` через механізм вспливання.

Щоб відмінити вспливання події, ви можете використовувати метод `stopPropagation()` на об'єкті події. Це призведе до того, що подія перестане "вспливати" далі по ієрархії елементів.

button.addEventListener("click", function(event) {

console.log("Кнопка була натиснута");

event.stopPropagation(); // Відмінити вспливання події

});

Важливо пам'ятати, що відмінюючи вспливання події, ви також можливо впливаєте на будь-які інші обробники подій, які можуть бути додані до батьківських елементів. Також варто враховувати, що в мережі елементів, де один елемент міститься в іншому, відмінення вспливання на одному з елементів може вплинути на можливість коректної роботи обробників подій на інших елементах.

1. Яка різниця між target та currentTarget в об'єкті події?

`target` та `currentTarget` - це дві властивості об'єкта події в JavaScript, які використовуються для отримання інформації про елементи, які спричинили виникнення події та які її обробляють. Однак вони можуть поводитися різним чином, коли виникає вспливання події.

1. `target`:

- Властивість `target` містить посилання на елемент, який спричинив виникнення події. Це означає, що `target` вказує на самий "цільовий" елемент, на який була спрямована подія.

- На практиці, якщо у вас є вкладений елемент, наприклад, кнопка всередині блоку, і ви клікнули на кнопку, то `target` буде посиланням на кнопку.

- Особливо важливо, що `target` залишається тим же, навіть якщо подія вспливає вгору по ієрархії елементів.

2. `currentTarget`:

- Властивість `currentTarget` містить посилання на елемент, на якому був встановлений обробник події. Це означає, що `currentTarget` вказує на елемент, на якому саме відбувається викликання обробника.

- Якщо обробник події доданий до батьківського елемента, а виникнення події спричинене дочірнім елементом, то `currentTarget` буде посиланням на батьківський елемент.

Ця різниця особливо важлива під час використання вспливання подій, коли подія може виникнути на дочірньому елементі, але обробник спрацьовує на батьківському елементі. В цьому випадку `target` вказує на дочірній елемент, який спричинив виникнення події, а `currentTarget` вказує на батьківський елемент, на якому відбувається викликання обробника.

1. Як отримати список елементів форми?

Для отримання списку елементів форми ви можете використовувати властивість `elements` форми, яка представляє колекцію всіх елементів форми. Ця колекція містить всі введення, кнопки, вибори тощо, які знаходяться всередині форми. Ось як ви можете це зробити:

<form id="myForm">

<input type="text" name="username">

<input type="email" name="email">

<input type="submit" value="Відправити">

</form>

const form = document.getElementById("myForm");

// Отримання списку елементів форми

const formElements = form.elements;

// Це буде HTML колекція, яку можна перетворити на масив для зручності

const formElementsArray = Array.from(formElements);

// Або використовуйте деструктуризацію для отримання масиву елементів

const [...formElementsArray] = formElements;

// Тепер у вас є масив елементів форми, з якими ви можете працювати

console.log(formElementsArray);

Це дозволить вам отримати всі елементи форми, які знаходяться всередині вказаної форми, і подальше взаємодіяти з ними, наприклад, отримувати значення, встановлювати атрибути тощо.

1. На які елементи можна додати події: focus та blur?

Події `focus` та `blur` відносяться до подій, пов'язаних з фокусуванням та зняттям фокусу з елементів у веб-документі. Ось на які елементи їх можна застосовувати:

1. `focus`:

Подія `focus` виникає, коли елемент отримує фокус, тобто стає активним та готовим приймати введення від користувача. Цю подію можна застосовувати до елементів, які можуть отримати фокус, таких як введення тексту, вибори, кнопки тощо.

Приклади елементів, до яких можна додати подію `focus`:

- `<input type="text">`

- `<textarea>`

- `<select>`

- Інші елементи, які призначені для введення даних.

2. `blur`:

Подія `blur` виникає, коли елемент втрачає фокус, тобто більше не є активним для введення від користувача. Цю подію також можна застосовувати до елементів, які можуть отримувати фокус.

Приклади елементів, до яких можна додати подію `blur`:

- `<input type="text">`

- `<textarea>`

- `<select>`

- Інші елементи, які призначені для введення даних.

Основна ідея цих подій полягає в взаємодії з користувачем: `focus` виникає, коли користувач "наводить" фокус на елемент, і `blur` виникає, коли користувач переходить до іншого елемента, виводить вміст елементу з фокусу або в інший спосіб призводить до втрати активності елементу.

1. Якими способами можна відправити форму (submit)?

Існує декілька способів, за допомогою яких можна відправити форму (тобто викликати подію `submit`). Ось декілька з них:

1. Використання кнопки "Submit":

Найпоширеніший спосіб - це використовувати кнопку з атрибутом `type="submit"`. Після натискання на цю кнопку відбувається відправлення форми.

<form id="myForm">

<!-- Формові поля -->

<button type="submit">Відправити</button>

</form>

2. Використання JavaScript:

Ви можете відправити форму за допомогою JavaScript, викликавши метод `submit()` об'єкта форми.

<form id="myForm">

<!-- Формові поля -->

<button type="button" id="submitButton">Відправити</button>

</form>

<script>

const submitButton = document.getElementById("submitButton");

const form = document.getElementById("myForm");

submitButton.addEventListener("click", function() {

form.submit();

});

</script>

3. Використання клавіші Enter:

Коли фокус знаходиться в одному з полів вводу форми, натискання клавіші Enter також викличе відправку форми.

1. Як змінити стандартну поведінку форми (відправку даних на сервер)?

Щоб змінити стандартну поведінку форми (відправку даних на сервер), ви можете використовувати обробник події `submit` і метод `preventDefault()` об'єкта події. Це дозволить вам перехопити відправку форми та виконати власні дії, наприклад, перевірити введені дані, виконати AJAX-запит або інші дії перед фактичною відправкою форми на сервер.

Ось приклад, як ви можете це зробити:

<form id="myForm">

<input type="text" name="username">

<input type="email" name="email">

<button type="submit">Відправити</button>

</form>

<script>

const form = document.getElementById("myForm");

form.addEventListener("submit", function(event) {

// Зупинити стандартну відправку форми

event.preventDefault();

// Ваші дії тут, наприклад, валідація даних, AJAX-запит і т.д.

// Після виконання необхідних дій можна вручну відправити форму

// (якщо це необхідно)

// form.submit();

});

</script>

У цьому прикладі, коли користувач натискає кнопку "Відправити", обробник події `submit` викликається. Метод `preventDefault()` перешкоджає стандартній відправці форми. Ви можете додати власний код для обробки даних або взаємодії з сервером. Після виконання необхідних дій, ви можете вручну відправити форму, викликавши метод `submit()` форми, якщо це необхідно.